2002

Benchmarking Java and .NET

J. Travis Schlist

Benchmarking Java and .NET

by

J. Travis Schlist

4/19/02

Southern Scholars Senior Project

With the advent of .NET there are now two major enterprise development environments, J2EE and .NET. Both use an intermediate language/virtual machine paradigm, which provides many capabilities and features that previous development environments did not. In deciding which platform to develop on there are many differences between the two that need to be taken into consideration, but one of the most important ones is speed. Which compiler/runtime environment will run my program faster? How much faster?

Measuring the speed of a system can be rather difficult, mainly because it begs the question "speed of what?" Do you want to know how many integer multiplications the machine can do in a second? How long memory accesses take? How much method overhead there is? How fast array operations are performed? Ultimately, the question the developer wants answered is, "how fast will MY program run?" which is a combination of all these factors.

My objective however is not to determine the relative performance for a specific type of application, but rather the performance characteristics of the two environments in general. To do this I will be measuring the speed of some of the basic operations, such as addition, subtraction, object creation, etc, and then testing some slightly larger benchmarks that use more operations, and then finally running some full scale applications. Knowing the speed of the basic operations can be nice if you know, for example, that your program is going to be doing a lot of floating point multiplications, but unfortunately it is hard to translate these speeds into actual performance differences since no real program uses floating point multiplication exclusively, except for perhaps the benchmark. The larger benchmarks can give a much better idea of how much faster a

real program will be, but unfortunately because of the complexity of the program, it is usually rather difficult to determine why. Used together however, they can be used to give a fairly good idea of the performance of each environment.

When measuring a speed characteristic, care also needs to be taken to ensure that the characteristic under examination is actually what the benchmark is measuring. For instance, when trying to measure the speed of memory accesses, one must be careful that the loop overhead, of the loop the memory accesses are running in, isn't so large that the speed of the memory accesses is insignificant. If this is the case then it is not the speed of memory accesses being measured, but rather the loop overhead. But as it turns out, when measuring the relative speeds of the two environments we do not really have to worry about this problem. Since the characteristic I am measuring is just the speed characteristics of the "virtual machine", I can use accepted benchmarks that measure normal machine performance and apply them to the virtual machine.

Thus, the main restriction on what benchmarks I could use was that I needed an implementation of the same benchmark for both environments. However, to make things simpler and to eliminate possible speed differences because of better/worse implementations of the benchmarks in different languages, I decided to use J#, the .NET version of Java, so that I could compile the same Java code onto both platforms. So I had to use benchmarks that provided source code. This was difficult since the majority of the Java benchmarks available only provide compiled class files, not source code.

The majority of the Java benchmarks I found that provided source code are what are called "synthetic" benchmarks (Grace 3). These are the small benchmarks that measure the speeds of basic operations and of simple tasks, like solving systems of linear

equations, performing LU factorization, etc. The Edinburgh Parallel Computing Centre[1] (EPCC), in association with the Java Grande Forum[2], has developed an extensive benchmark suite of this type, the Java Grande Benchmark Suite[3]. This suite measures most of the basic operations as well as providing many of the slightly larger benchmarks. It is this suite I will be using for my low-level benchmarks.

The Java Grande benchmarks are split up into 3 sections. The first section is comprised of benchmarks that "measure the performance of low level operations such as arithmetic and maths library operations, method calls and casting" (JavaG). The second section is made up of what are called "kernels", "short codes which carry out specific operations frequently used in" (JavaG) scientific applications. Some of these benchmarks include Fast Fourier Transformations, IDEA encryption, and Fourier coefficient analysis. The third section is made up of real, large-scale scientific applications, approaching those of the SPEC, like a molecular dynamics simulation, a Monte Carlo simulation and an alpha-beta pruned search, which solves a game of connect-4. The second and third sections allow the benchmarks to be run on different data sizes to give an idea of the scalability of the Java virtual machines (JVM) being tested.

For large-scale benchmarks I was much more limited. There are very few large-scale benchmarks written in Java, and of the few I was able to find only a fraction provided source code. The Standard Performance Evaluation Corporation's[4] (SPEC) JVM98 benchmark suite provides source code for several of its benchmarks, and is held in high regard by the computing community, so it is the benchmark suite I decided to use

---

[1] www.epcc.org
[2] www.javagrande.org
[3] www.epcc.ed.ac.uk/computing/research_activities/java_grande/index_1.html
[4] www.spec.org

for my large-scale applications.

The SPEC JVM98 benchmark suite consists of 7 real-world applications that have been adapted as benchmarks and are specifically designed for comparing the speed of Java virtual machines. They should give a better idea of the relative speeds of the systems. Unfortunately, 3 of these benchmarks do not come with source code, so I was only able to use 4 of them. The first of these benchmarks is _201_compress, which uses a modified Lempel-Ziv method to compress a large amount of data. The second is _202_jess, the Java Expert Shell System, based on NASA's CLIPS expert shell. "In simplest terms, an expert shell system continuously applies a set of if-then statements, called rules, to a set of data, called the fact list" (SPEC). The third benchmark, _209_db, performs multiple database operations on a memory resident database. The fourth and final benchmark I was able to perform was _227_mtrt, "a raytracer that works on a scene depicting a dinosaur, where two threads each renders the scene in the input file time-test model, which is 340KB in size" (SPEC). Another application is included, _200_check, but it only checks the capabilities of the system to ensure it can run the JVM98 benchmarks and does not calculate any real performance measurements.

It is important to note that I had to break the SPEC run and reporting rules in running the benchmarks, namely, I recompiled the source files and ran the benchmarks in a console instead of an applet. However, I did this to make the runtime conditions of .NET and Java as similar as possible, so while these results are not reportable, they should give a good idea of the relative speed of the environments.

In my comparison I decided to use 3 of the most popular Java virtual machines to represent the Java side of the battle. First of all I decided to use the newest offering from

Sun, JRE1.4.0. I also included the latest IBM virtual machine, 1.3.0, which in the past has been the leading JVM performer, and also the latest Microsoft Java runtime, jview, build 3804, another solid virtual machine.

To compile the Java files I used Sun's JDK 1.3.1 compiler. I would have liked to use Sun's newest compiler, 1.4.0, but IBM has not yet release a Java 1.4.0 compatible virtual machine, so I was restricted to version 1.3.1. I compiled the Java Grande benchmarks as described in the documentation without any problems, but the SPEC benchmarks required a little more work. I had to make several modifications to the code to get them to compile under both environments, but all of them were cosmetic and did not change the functionality of the benchmarks. I then made scripts to run all of the Java Grande benchmarks for each of the 3 JVM's. The only significant flags I used in compiling and running the benchmarks were the –noverify flag to get the Section 1 Serial benchmark running on the IBM and Sun JVM's, and the –mx flag to increase the maximum amount of memory available to the virtual machine on certain benchmarks that required it.

On the .NET side of things I compiled the code from the command line with the J# beta 2 compiler, and ran the benchmarks under the .NET Framework, service pack 1 with the J# redistributable installed. To compile the Java Grande benchmarks I first made a dll of the utilities in the jgfutil directory using the /target:library flag and then copied this dll to each of the section directories. I then compiled each of the benchmarks in the sections, using the /r: flag to reference the dll. No other significant flags were used in the compilation or running of the benchmarks. The /o optimization flag was NOT used. To compile the SPEC benchmarks I just used the /main:SpecApplication and the

/recurse:*.java flags to compile all the Java files into a .NET executable.

All the benchmarks were run on three different machines, a PIII 800, an AMD 1400 and a P4 1800, all running Windows 2000 sp2 with 256mb of memory.

There were also a couple Java Grande benchmarks that never completed on certain environments when I ran them so are not included in the results. These include the FFT benchmarks in section 2, size B and C, and the Monte Carlo benchmark size B in the third section. I cannot give a good reason for why they never finished, but in looking through the results published on the EPCC website I found that nearly every Intel based system that ran these benchmarks, with the exception of dual processors, left these results blank as well, so I am not the only one to experience problems with these specific benchmarks. Most of them did publish results for the FFT size B benchmark, but my problem with this benchmark is probably just an extension of this same problem.

In discussing the results of the Java Grande benchmarks, all calculated results are given in JGF Numbers. These numbers are found by first finding the ratio of the calculated results to the results from a reference machine, and then taking the geometric mean of these calculated ratios. This effectively calculates how many times faster the environment being tested is than the reference machine (larger numbers are better). However, one benchmark each from 3 of the sections was not used, so the summary JGF Numbers for these sections are not valid JGF Numbers, and cannot be compared to other published results. The missing benchmarks would skew the comparison.

The results of the benchmarks were somewhat mixed. As shown in figure 1, the IBM JVM came out far ahead of the others on the basic operations tests of section 1 in the Java Grande benchmarks. However, this is due in large part to the ridiculously high

results of a couple benchmarks in this section, and one benchmark in particular, the Create benchmark, which measures how fast the VM can allocate primitives and objects. This astronomical result, a whopping 230 on the AMD 1400, compared to 10 on the Microsoft VM, is almost certainly an optimization of the Just In Time (JIT) compiler, which means that instead of actually creating the object like the benchmark expects, the compiler realized that the object was not being used and just did not create it. This means that the amount of time it actually takes to create an object has not decreased, only that the VM is good at figuring out when it does not have to create the object, which happens a lot during a benchmark but is relatively rare in real situations, so will translate into only marginal performance gains at best.

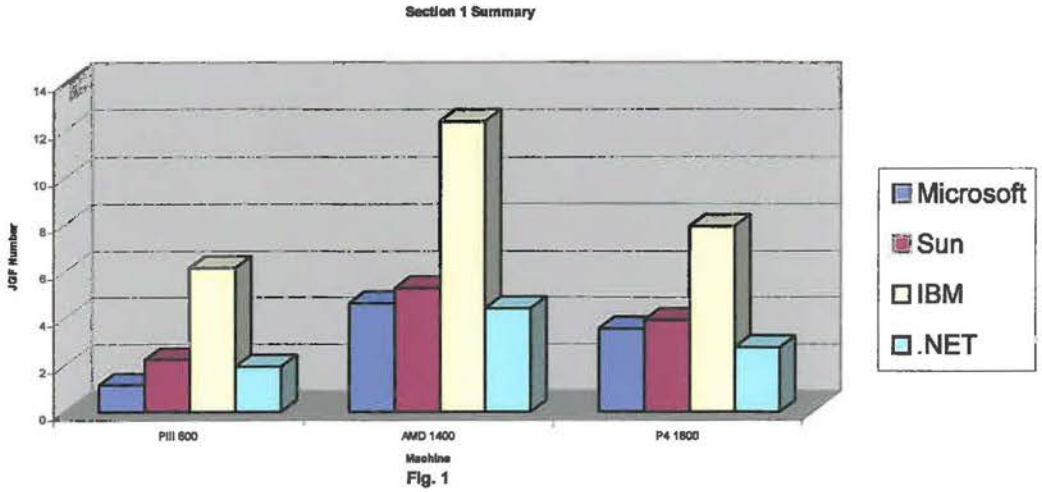

Fig. 1

The results of the section 2 benchmarks were a lot more even. Overall Java, with the Microsoft and IBM JVM's, came out neck and neck with the .NET runtime. The notable exception to this was the Microsoft JVM running on the PIII 800. For some reason the Microsoft JVM seemed to do poorly on that machine throughout all my tests. The only real loser though on this set of benchmarks was Sun's JVM, which came out well below the others.

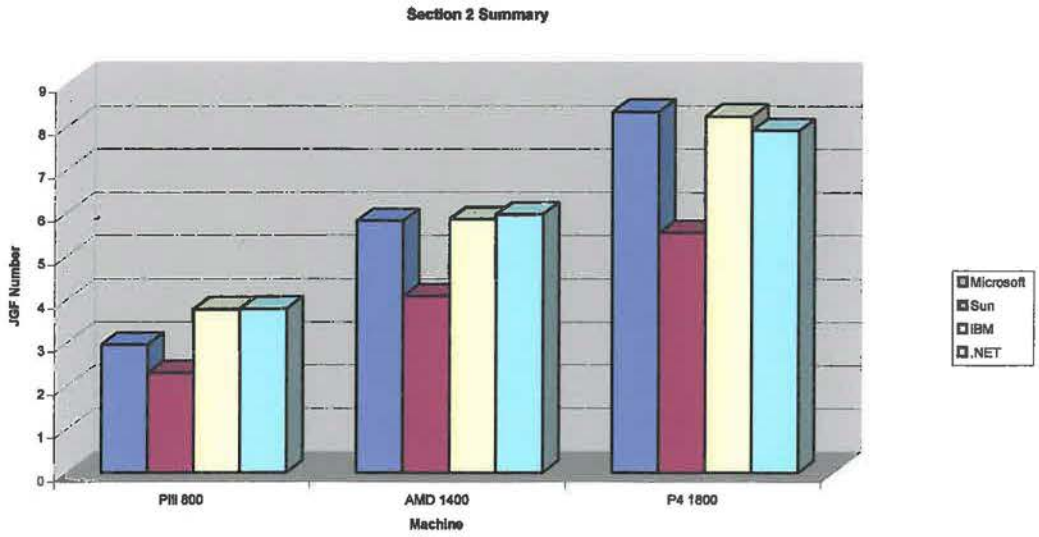

Fig. 2

In section 3 of the Java Grande benchmarks the roles were reversed on the Java side of things. On this set of benchmarks Sun really shined, consistently beating the other two JVMs by a fair margin. However, .NET stayed just a little bit ahead of it. Overall, .NET performed very well on the Java Grande benchmarks. With the exception of the section 1 benchmarks, which are not representative of true performance anyway, .NET was consistently at the top, or very close to it.

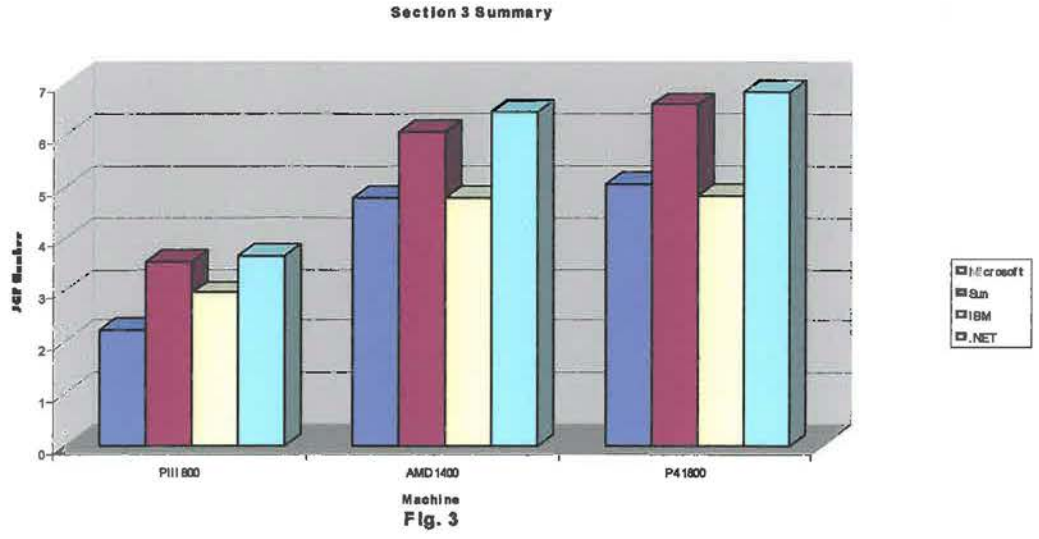

Fig. 3

The results of the SPEC benchmarks were quite different. With the exception of the compress benchmark, the IBM and Sun JVMs beat .NET consistently, as shown in figure 4 (larger numbers are better). In many of the tests the Microsoft JVM even beat .NET, but Microsoft's poor relative performance on the PIII 800 dragged its averages below that of .NET, as shown in figure 5, which displays the geometric means of the benchmarks for each system.

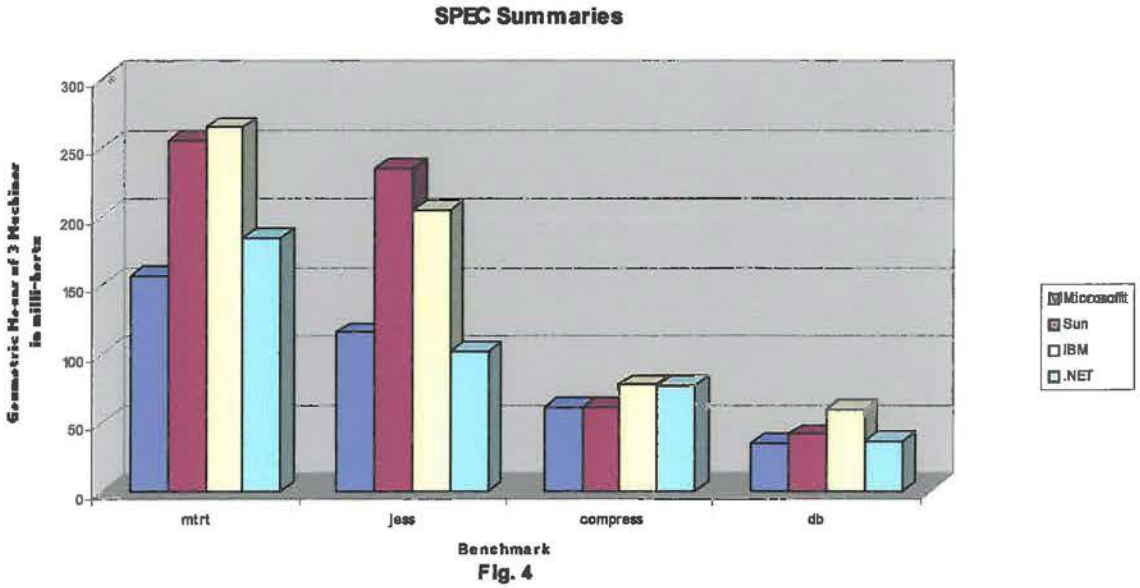

Fig. 4

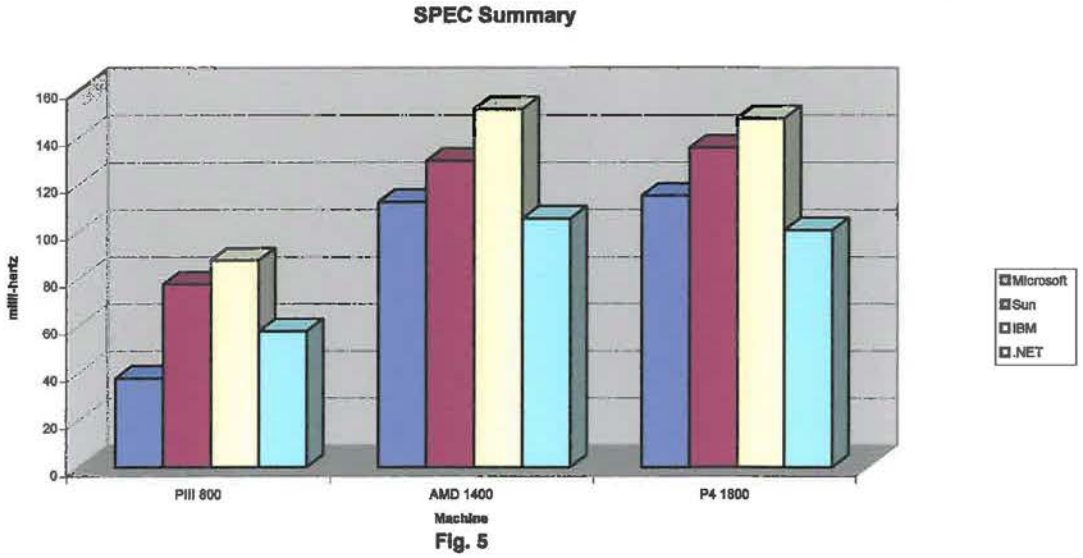

Fig. 5

The results from these benchmarks are too mixed to draw any hard conclusions from. The mixed results can be explained by the fact that the different benchmarks are measuring different aspects of the environment, but an in-depth analysis of what exactly each benchmark is measuring, and why .NET performed better on certain benchmarks than on others would be beyond the scope of this preliminary investigation. Overall, .NET seems to be a solid performer, and while it appears to be slightly slower on average than Java, it is still a very immature environment with plenty of opportunity for growth, and the J# compiler I used was still in beta. I would look for .NET to be a very competitive performer in the future.

Bibliography

Grace, Rich. The Benchmark Book. Upper Saddle River: Prentice Hall P T R, 1996

"JavaG Benchmarking"
<http://www.epcc.ed.ac.uk/computing/research_activities/java_grande/seq/contents.html> (4/19/02)

"SPEC JVM Client98 Help"
<http://www.spec.org/osg/jvm98/jvm98/doc/benchmarks/index.html> (4/19/02)

SOUTHERN SCHOLARS SENIOR PROJECT

Name: Jonathon Travis Schlisf Date: 1/30/02 Major: Computer Science / Mathematics

SENIOR PROJECT

A significant scholarly project, involving research, writing, or special performance, appropriate to the major in question, is ordinarily completed the senior year. The project is expected to be of sufficiently high quality to warrant a grade of A and to justify public presentation.

Under the guidance of a faculty advisor, the Senior Project should be an original work, should use primary sources when applicable, should have a table of contents and works cited page, should give convincing evidence to support a strong thesis, and should use the methods and writing style appropriate to the discipline.

The completed project, to be turned in in duplicate, must be approved by the Honors Committee in consultation with the student's supervising professor three weeks prior to graduation. Please include the advisor's name on the title page. The 2-3 hours of credit for this project is done as directed study or in a research class.

Keeping in mind the above senior project description, please describe in as much detail as you can the project you will undertake. You may attach a separate sheet if you wish:

My project will be a comparison of the C# programming language in .NET to Java and its latest runtime environment. I will first do research on what ~~[illegible]~~ is required for a good benchmark, and then using either benchmarks I develop or already existing benchmarks I will perform a battery of tests to determine the speed advantages/disadvantages of each language and their run-time environment.

Signature of faculty advisor [signature] Expected date of completion _____

Approval to be signed by faculty advisor when completed:

This project has been completed as planned: ✓

This in an "A" project: ✓

This project is worth 2-3 hours of credit: ✓

Advisor's Final Signature [signature]

Chair, Honors Committee _____ Date Approved: _____

Dear Advisor, please write your final evaluation on the project on the reverse side of this page. Comment on the characteristics that make this "A" quality work.

Travis did an outstanding job on this project. Benchmarking is not an easy task and requires lot of time and preparation. The effort required to produce data for the graphs he gives in his paper is significant. The numbe of benchmark programs used is exceptional. Although the results he got were not what he might have expected they demonstrated the difficulty of benchmarking. Explanations given for the results observed demonstrate excellent understanding of those problems.